(19) Europäisches Patentamt
European Patent Office
Office eur péen des brevets

(11) Publication number: **0 365 188 A2**

(12) EUROPEAN PATENT APPLICATION

(21) Application number: **89310271.5**

(51) Int. Cl.5: **G06F 9/38**

(22) Date of filing: **06.10.89**

(30) Priority: **18.10.88 US 259345**

(43) Date of publication of application:
25.04.90 Bulletin 90/17

(84) Designated Contracting States:
AT BE CH DE ES FR GB GR IT LI LU NL SE

(71) Applicant: **APOLLO COMPUTER INC.**
33 Billerica Road
Chelmsford Massachusetts 01824(US)

(72) Inventor: **Barbour, Russell Gray**
67 Carter Drive
Framingham Massachusetts 01701(US)
Inventor: **Soeder, Carl A.**
166 Pine Hill Road
Boxborough Massachusetts 01719(US)
Inventor: **Ciavaglia, Stephen J.**
23 Macdonald Drive
Nashua New Hampshire 03062(US)

(74) Representative: **Arthur, Bryan Edward et al**
4 Dyers Buildings Holborn
London, EC1N 2JT(GB)

(54) Central processor condition code method and apparatus.

(57) A method and apparatus is disclosed for control of a central processor in response to a branch instruction using two separate, subsequently updated condition codes. Computer architecture is provided wherein the condition codes which determine the processor state result from the execution of instructions prior to the currently executing instruction. When the preceding instructions are executed, condition codes are set and maintained in a first condition code register. The first condition code is transferred to the second condition code register, and the first condition code register is updated to reflect the result of the current instruction execution. Any condition code state such as a branch used by the third instruction is based on the condition code state maintained in the second condition code register. The processor is provided with code which compiled according to the present invention, provides improved processor performance by reducing delays in instruction execution. If the current instruction is a condition code dependant instruction such as a branch instruction, this third instruction will execute based on the condition code maintained in the second condition code register.

EP 0 365 188 A2

FIELD OF THE INVENTION

The present invention relates to computer instruction execution and compiling systems, and in particular to computer processor and compiling system techniques which relate to the latency instructions occurring between the establishment of a condition code state and the execution of the subsequent condition code dependant instruction.

BACKGROUND OF THE INVENTION

In computer architectures, the result of an execution of a given instruction causes the processor to maintain a particular state. A summary of the result of a given operation is stored as a condition code, and the state is referred to as a condition code state. Changes in program flow, such as caused by a conditional branch, are achieved by testing and branching on a particular condition code state. However, many computer systems suffer a settling delay between the setting of the condition codes to the use of those condition codes by a conditional branch instruction. In some computer systems, time is wasted by either increasing the machine's cycle time or the injection of a system stall. Either approach requires the branch instruction to wait for the condition codes to stabilize reducing system performance. Pipeline processors have instructions which are partitioned into several cycles of execution, each cycle of which is completed at different and sequential time periods, and have the cycles of the instruction partially overlapping to allow the most efficient execution of all such pipelined instructions. Pipeline processors are particularly penalized when the program flow is altered by a branch instruction. As a technique to reduce this penalty, the step of branching on condition code state by instruction (i + 1) is overlapped with the step of instruction execution by the prior instruction, i. However, this action leads to a condition code settling delay of one cycle in that instruction (i + 1) not alter condition code state while the condition codes becomes stable for use in instruction i + 2. Condition code settling imposes the restriction that there is a single instruction delay (referred to as the later instruction) between an instruction which alters condition code state relative to an instruction which uses that updated state. This restriction restrains the compiler from assigning instructions as latency instructions. As a result, processor performance is reduced since useful latency instructions cannot be assigned.

SUMMARY OF THE INVENTION

The present invention provides an apparatus and method to maintain two sets of condition codes which allow the compiler to assign any instruction as a latency instruction, filling the intervening latency time periods with instructions which may set or use condition codes, thereby providing the maximum improvement in processor performance with pipelined instructions.

In particular embodiment of the invention discussed herein, it is assumed, that the number of latency instructions is 1. According to the present invention, the architecture supports two sets of condition codes referred to as Current cc and Next cc. The same state variables are represented in both sets of condition codes. All branching is relative to the condition codes held in Current cc. The Current cc is always updated from Next cc, which may be updated from the executing instruction at the option of that instruction. As the execution of instruction i is completed, the state maintained in Next cc is transferred to Current cc, and the state maintained in Next cc is selectively updated to reflect current instruction execution. If an instruction does not then alter the condition code, Next cc remains unchanged, forcing the current cc state to be equal to the Next cc state. As the latency instruction, or subsequent instruction i + 1 completes, the state maintained in Next cc is transferred to Current cc, and again the state maintained in Next cc is updated to reflect the present executing instruction. If the next, or third instruction, (i + 2) is a branch or other instruction causing non-sequential instruction execution, this branch is based on the condition codes maintained in Current cc.

Therefore, the apparatus and method of the present invention provides increased efficiency wherein the pipeline processor altered state by instruction i, may also be altered by the latency instruction i + 1,

affecting the execution of instruction i + 2 and i + 3 respectively, if such instructions are condition code state dependant, resulting in enhanced system performance by making the otherwise useless lat ncy instructions available for a larger s t of operations.

BRIEF DESCRIPTION OF THE DRAWING

These and other features according to the present invention will be better understood by reading the following detailed description, taken together with the drawing wherein:

Fig. 1 is a block diagram of one embodiment of the apparatus according to the present invention;

Fig. 2 is a more detailed diagram of the condition code pipeline of the embodiment of Fig. 1; and

Fig. 3 is a more detailed diagram of the process control unit according to the embodiment of Fig. 1.

DETAILED DESCRIPTION OF THE INVENTION

The system according to the present invention provides a pipeline structure for one or both integer and floating point condition codes. The branch operation predicate is partitioned into a integer and floating point branch condition portion. In the particular embodiment shown, the integer processor is responsible for interpreting both integer and floating point branch predicates and generates the integer condition codes external to the Integer Processor. A copy of this maintained state is sent and held in the Integer Processor.

The instructions received by the processor in Fig 1 are provided by an instruction cache 102 during the Instruction Fetch (IF) period to this instruction decode register 104 in response to the cache address signal, PC SRC provided by a program counter 130. The File Register 112 of the Execution Unit 110 receives the instruction from the instruction decode register 104 and provides Operand A and Operand B in response to the received instruction during the Operand Fetch (OF) period. Operands A and B are received and stored in registers 114 and 116 at substantially the same time the execute register 106 receives and stores the instruction stored in the instruction decode register 104. The Arithmetic Logic Unit (ALU) 118 processes Operands A and B, and provides a result and a condition code signals during the Instruction Execute (IE) period which signals are stored in registers 120 and 122 respectively. Substantially simultaneously with the storage of signals from the ALU 118 in registers 120 and 122, the instruction stored in the update register 108. The instruction instruction stored in the update register 108. The instruction stored in the update register may be received by the File Register 112 to indicate where to store the signal now stored in the result register 120, as determined by the particular instruction executed during the Writeback Execute (WE) and data Cache (DC) period. Alternately, results may be stored during a subsequent Writeback Load (WL) period.

The program counter 130 includes branch select logic 132 which controls the operation of the PC Source Selector 134 in response to the selected condition code (126) and the instruction (104) being decoded for execution by the File register 112. The program counter register 136 can be incremented by zero incremented by a value corresponding to the address difference (e. g., 8 provided by 138) in the next instruction in the cache 102.

Each condition code includes at least four single bit integer condition codes (cc), including zero (Z), negative (N), overflow (V), and carry (C). In the particular embodiment of the present invention, there are six floating point codes not-a-number (NAN), zero (Z), negative (N), (i) infinity, graphic trivial accept and graphic trivial reject. The predicate field of the floating point branch will support up to 32 floating point branch types such as illustrated in copending patent application APOLL-108XX, entitled APPARATUS FOR SELECTIVE EXECUTION OF INSTRUCTIONS FOLLOWING A BRANCH INSTRUCTION, filed concurrently herewith, incorporated by reference. The location of the condition codes in the processor status word (IPSW) is shown in table 1, below. The system according to the present invention also includes a compiler which unconditionally inserts at least one instruction between an instruction which sets condition codes and an instruction which is a branch (or other instruction causing non-sequated instruction execution) based on those condition codes to provide time for condition code s ttling. The inserted instruction can also set the condition codes for reference by subsequent branch (test) instructions or use the pr viously set condition codes. Condition Code state is maintained in a register referred to as the Integer Processor Status Longwood register (IPSW).

The position of condition codes in a processor status longword (IPSW) is illustrated in table 1 below.

TABLE 1

IPSW FORMAT

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14		00
N	Z	V	C	N	Z	V	C										R	'0'	
current				next															
cc																			cc

Z = zero (r/w)
V = overflow (r/w)
N = negative (r/w)
C = carry (r/w)
R = completed in shadow of load cc

There are three classes to instructions which can set integer condition codes:

1) Register to register instruction which sets condition codes (rr.cc) where the execution unit specifies the condition codes based on computed results.

2) Integer load instruction which sets condition codes (load.cc) where the condition codes are generated on an incoming result supplied by the data cache interface. A load.cc takes one additional cycle to generate condition codes relative to an rr.cc.

3) Move to processor register instruction which updates the condition code pipeline (mtpr.ipsw). A mtpr.ipsw takes on additional cycle to generate condition codes relative to an rr.cc.

The condition code pipeline shown in the block diagram 150 of Fig. 2 The current cc register 124 holds the integer condition codes relative to the (last instruction-1 executed, and is sourced from the current cc select multiplexor 160. The next cc register 122 holds the integer condition codes relative to the (last instruction) executed and is sourced from the next cc select multiplexer 162. The cc select multiplexor 166 supplies the branch condition codes to the program control unit 130 (Fig. 3) which are used to alter instruction stream direction and is selectively sourced from stet the current cc or next cc registers, 124 and 122. The next cc select multiplexor 162 is used to supply the source of integer condition codes to the next cc register 122 and is selectively sourced from the execution unit (110 of Fig. 1) load__cc 154 or mtpr__ipsw 156 registers. The current cc select multiplexor 160 is used to supply the source of integer condition cods to the current cc register 124 and is sourced from the next cc, load__cc or mtpr ipsw registers, 122, 154 and 156. The load__cc register 154 holds condition code state relative to load.cc instruction execution. The mtpr__ipsw register 156 holds condition code state relative to mtpr.ipsw instruction execution. The ipsw .register represents the state of the condition code pipeline (Current and Next CC Registers) and all necessary state in order to advance the pipeline relative to past instruction history.

In the description below, only one condition code update source S(execution unit, load__cc register or mtpr__ipsw register) can be injected into the condition code pipeline per machine cycle. However, other embodiments may permit multiple code update sources per machine cycle. IF an rr.cc follows a load.cc or mtpr__ipsw, instruction control hardware will stall the rr.cc by one cycle forcing a null cycle in the execution unit.

For register to register instruction execution, as instructions are executed the cc select multiplexor selects the next cc register to supply the branch condition codes with the next cc register conditionally loaded into the current cc register. The next cc register is updated from the executing unit provided there is a valid instruction currently being executed which sets integer condition codes (rr.cc). If there is nit, the next cc register will not be updated. The current cc register is updated from the n xt cc register if ther is a valid instruction being executed (which may or may not be setting condition cod s). All instructions are allocated a cycle to the execution unit but not all cycles are executing an instruction due to resource stalls, instruction cache misses, ect. If there is not a valid instruction in the execution unit, neither th current cc or next cc registers will be updated. The cc select multiplexer will select the next cc register.

For a load.cc instruction, the cc select multiplexor selects the next cc register to supply branch

conditions, the next cc register is updated from th load__cc register and the current cc register conditionally updated from the load__cc register. Since load.cc instruction takes an additional cycle to specify cc state, an additional instruction may have been executed by the execution unit. Condition code pipeline advance is therefore a function of whether an additional instruction has be n x cuted when the load__cc register is ready to supply condition code state via the load__cc register to the condition code pipeline. This history is maintained in the R bit of the IPSW. If an additional instruction has been executed by the execution unit (note this instruction cannot set condition codes), the current cc register is also updated from the load__cc register. The result is that both the current cc and next cc registers are updated from the load__cc register. Since two instructions have executed (the load.cc and it's shadow), the condition code pipeline must be advanced by two to reflect this once load.cc condition codes are available. If an additional instruction had not been executed, only the next cc register will be updated from the load__cc register.

For a mtpr.ipsw instruction, the cc select multiplexor selects the next cc register to supply branch conditions with the current cc and next cc registers unconditionally updated from the mtpr__ipsw register. That is the current cc register will be updated from the current cc portion of the IPSW and the next cc register will be updated from the next cc portion of the IPSW. The mtpr.ipsw instruction is used to overwrite the cc pipeline.

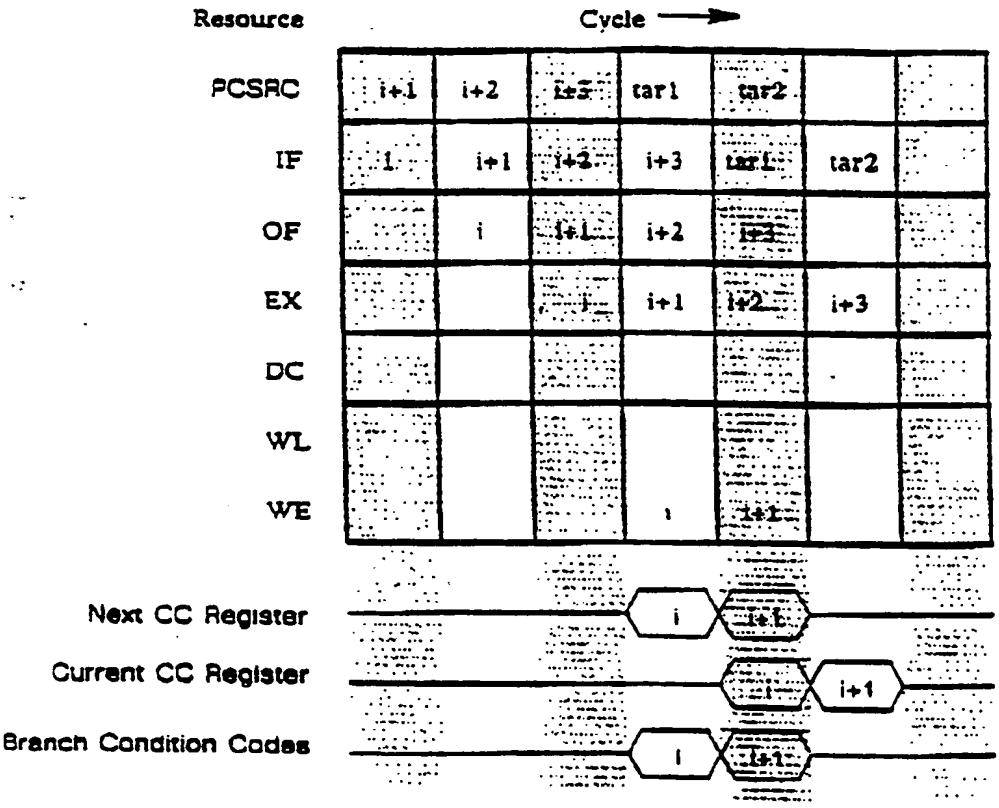
For the mtpr.ipsw instruction, the cc select multiplexor selects the current cc register to supply branch conditions with the current cc and next c registers unconditionally updated from the mtpr__ipsw register with IPSW state. That is the current cc register will be updated from the current cc portion of the IPSW and the next cc register will be updated from the next cc portion of the IPSW. The mtpr.ipsw instruction is used to overwrite the condition code pipeline.

At exception trat entry, the condition code pipeline will reflect the state of the last two instruction executed. If Instructions i-1 and 1 are the last two instructions executed, current cc register will reflect condition code state updated by instruction i-1 and the next cc register will reflect condition code state updated by instruction 1. During return from exception, the condition code pipeline is disabled from advancing. The condition code pipeline will not advance until the first instruction after returning from the exception has reached the execution unit. This equates to the second cycle following return from exception. During the first cycle from exception return, the cc select multiplexor will select the current cc register. During the second cycle from exception return, the cc select multiplexor will select the next cc register. From the third cycle on, the cc select multiplexor will select the next cc register.

Reservation tables (Tables 2-6) depicting the above action follow.

TABLE 2

RR.cc instruction execution...

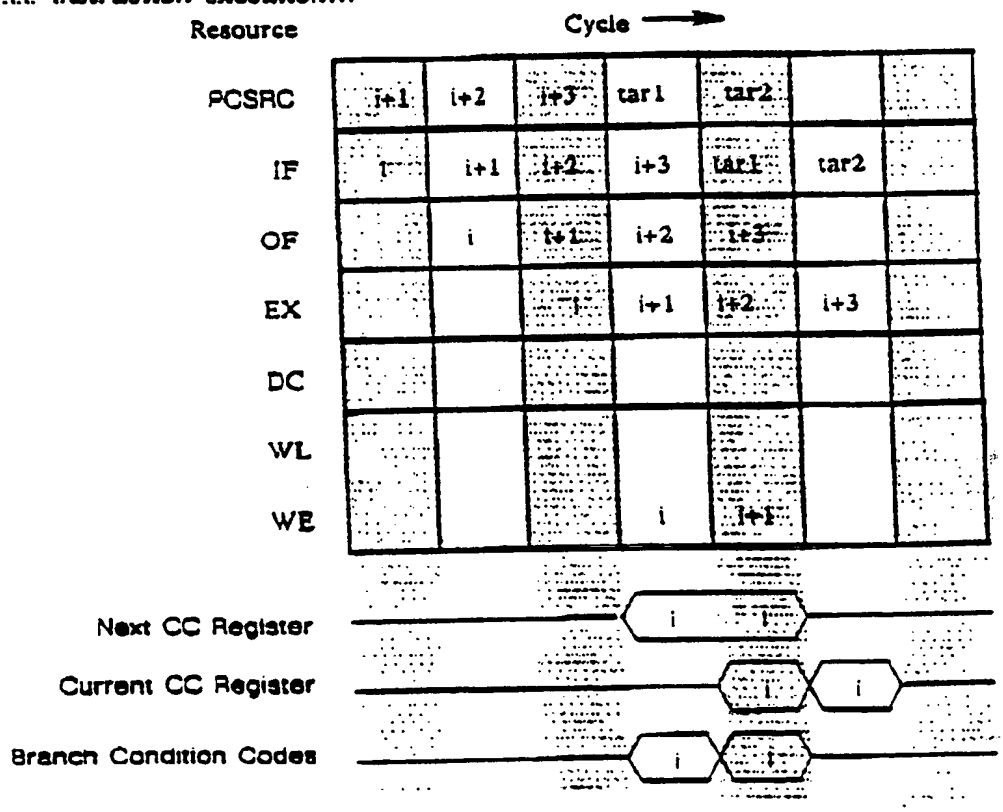


Code Sequence:

i: rr.cc
i+1: rr.cc
i+2: b<cond> tar1
i+3: b<cond> tar2

TABLE 3

RR/RR.cc instruction execution...

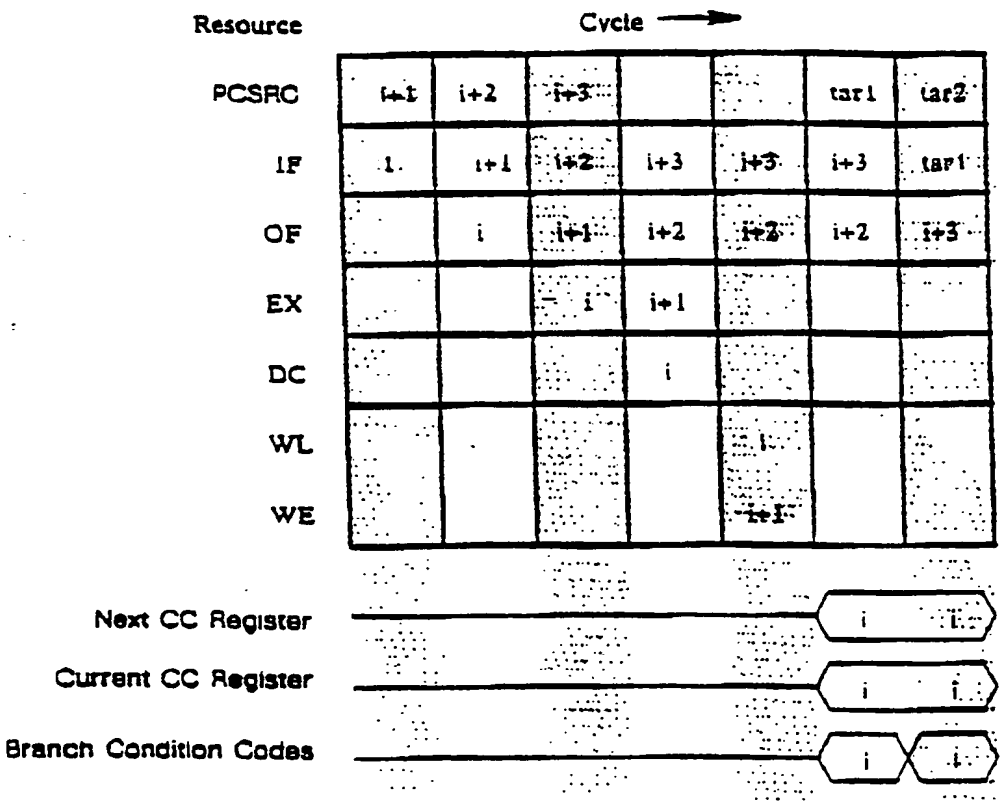


Code Sequence:

```
i:   rr.cc
i+1: rr
i+2: b<cond> tar1
i+3: b<cond> tar2
```

TABLE 4

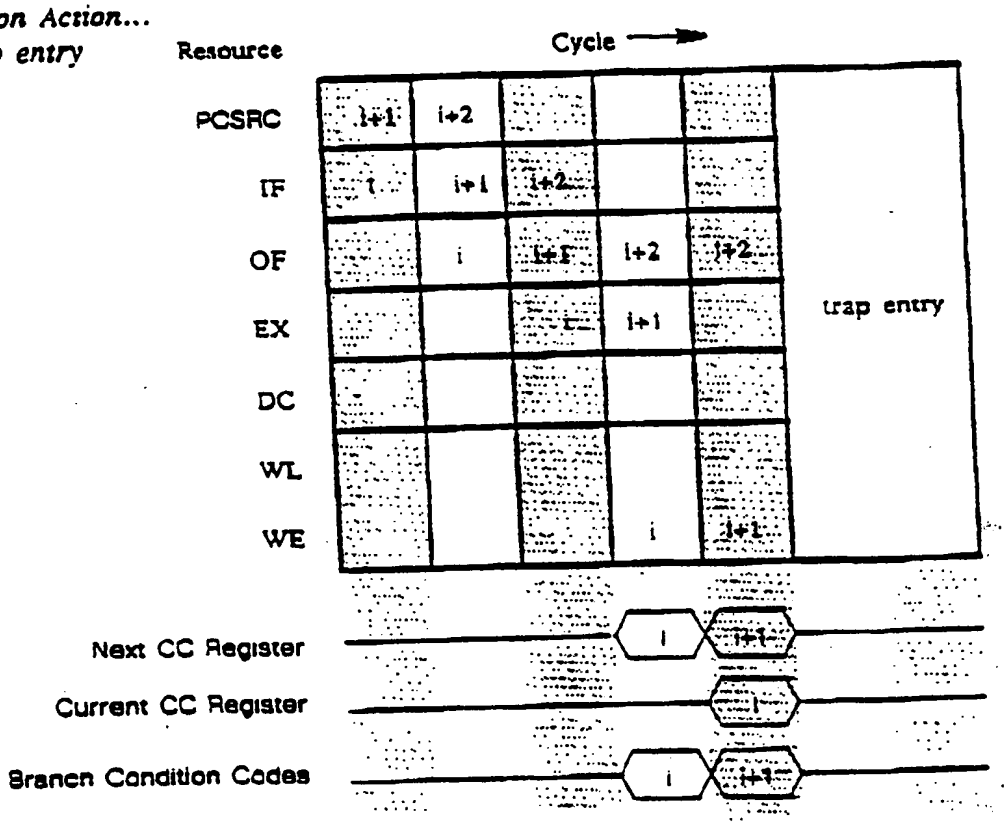
load.cc instruction execution...



Code Sequence:

```
i:   load.cc
i+1: rr
i+2: b<cond> tar1
i+3: b<cond> tar2
```

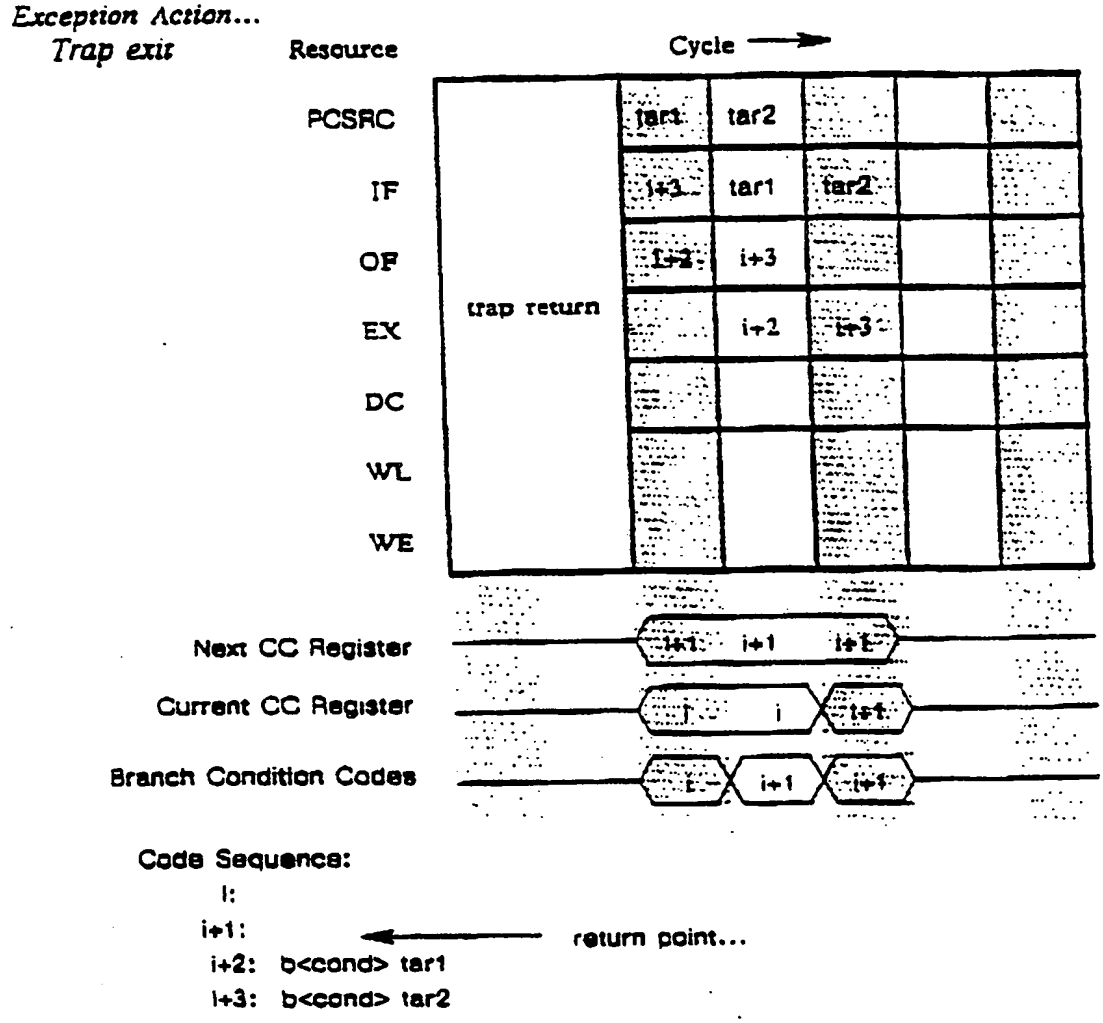
TABLE 5



Code Sequence:

```
i:   rr.cc
i+1: rr.cc   <-- trap here, i and i+1 complete...
i+2: b<cond> tar1
i+3: b<cond> tar2
```

TABLE 6



Reservation table definitions are as follows.

Instruction Fetch (IF)

The instruction Fetch stage is responsible for fetching the next instruction from the instruction Cache. During this cycle the IP will be supplying the virtual PC address of the subsequent instruction to be fetched.

Operand Fetch/Instruction Decode (OF)

The Operand Fetch and Instruction Decode stage is responsible for decoding the instruction and dispatching the necessary operands and control to the Execution Unit. Integer register file operands are fetched in this stage. For macro branch instructions, this stage supplies the branch target address or next sequential address of the next instruction to be fetched.

Instruction Execute (EX)

The Instruction Execute stage is responsible for xecuting the instruction. For memory r fer nce instructions, this stage computes the effective address.

Writeback Execution Result (WE)

The Writeback Execution Result stage writes the computed Execution Unit result back into the integer register file.

Data Cache Address (DE)

The Data Cache Access stage is responsible for accessing the data cache for memory reference instruction. For store operations, this stage transmits data to the Data Cache. For load operations, this stage receives data from the Data Cache. Only memory reference instructions have this stage.

Writeback Load Result (WL)

The Writeback Load Result stage is responsible for writing load data into the integer register file. Only memory reference instructions have this stage.

The instructions to be executed in the processor according to the present invention are provided from the instruction cache 102, a memory or other instruction store (not shown) according to a sequence of address signals, provided by a program counter source bus (PCSRC) 152 of the program control unit block diagram 130 of Fig. 3. The address signals on lead 152 is selectively provided by a program counter multiplexer 134A which selects the program counter signals from a variety of sources according to a control signal provided by a branch select logic element 158. When an instruction trap condition is invoked, a trap address is provided by a trap unit 160 and selected by a trap select signal 162. Similarly, branch type signals 164 cause the branch select logic 158 and the multiplexor 134A to select a signal from either Register file A Part 1 or from a 36 bit adder 176. The condition code pipeline 150 in Fig. 2 provides a branch condition code to branch generate control logic 174. The program counter multiplexer 134A selectively receives the address signal from an adder 176 which provides on address resulting from the sum of a displacement on the 178 and a decrementing program counter signal on the 180, which relates to the location of the destination of branch taken. By further example, after having taken the branch, the program counter address is stored by a fetch register 136 and conditionally incremented by an adder 138 upon receipt of an instruction register control signal to provide an incrementing program counter signal to the multiplexer 156.

Substitutions and modifications made by one of ordinary skill in the art are within the scope of the present invention. For instance, the addition of intervening steps within the process according to the present invention and the addition of registers in the instruction pipeline as illustrated in the structure according to the present invention is within the scope of the present invention. Furthermore, modification to the format of the instructions or codes described herein to provide the selective execution of branch shadow instructions is also considered to be within the scope of the present invention, which is not to be limited except by the claims which follow:

Claims

1. A programmable computer, comprising:
means for providing sequence program instructors according to a selectively applied sequence of address signals, wherein selected contiguous location in program storage means contain program instructions to b executed in sequence having at least one program instruction which enables a change in a condition code; and
said contiguous program instructions further include a condition code responsive instruction having an offset of at least two locations subsequent to said program instructions affecting said condition code; and
means for processing data according to said sequence of program instructions including condition code

responsiv m ans, wh rein said contiguous program instructions ar executed according to a predetermined address sequence and the state of said condition code responsive means.

2. A m thod of executing a programmabl sequence of instructions comprising the steps of:
providing a sequence of program instructions according to a selectively applied sequence of address signals, said sequence of program instructions comprising at least one program instruction which sets the state of a condition code then executed;
providing a condition code responsive instruction in said sequence of program instructions having an offset of at least two locations subsequent to said instructions affecting said condition code; and
executing said sequence of program instructions according to a predetermined sequence of address signals and the state of said condition code.

3. A method of compiling a computer program, comprising the steps of:
providing machine code instruction in a contiguous sequence according to a higher level program command;
providing machine code instructions which contend the state of condition codes in said contiguous sequence;
providing machine code instructions which are responsive to said condition codes at a sequential offset of at least two machine code instructions subsequent to said machine code instructions which control the state of condition codes.

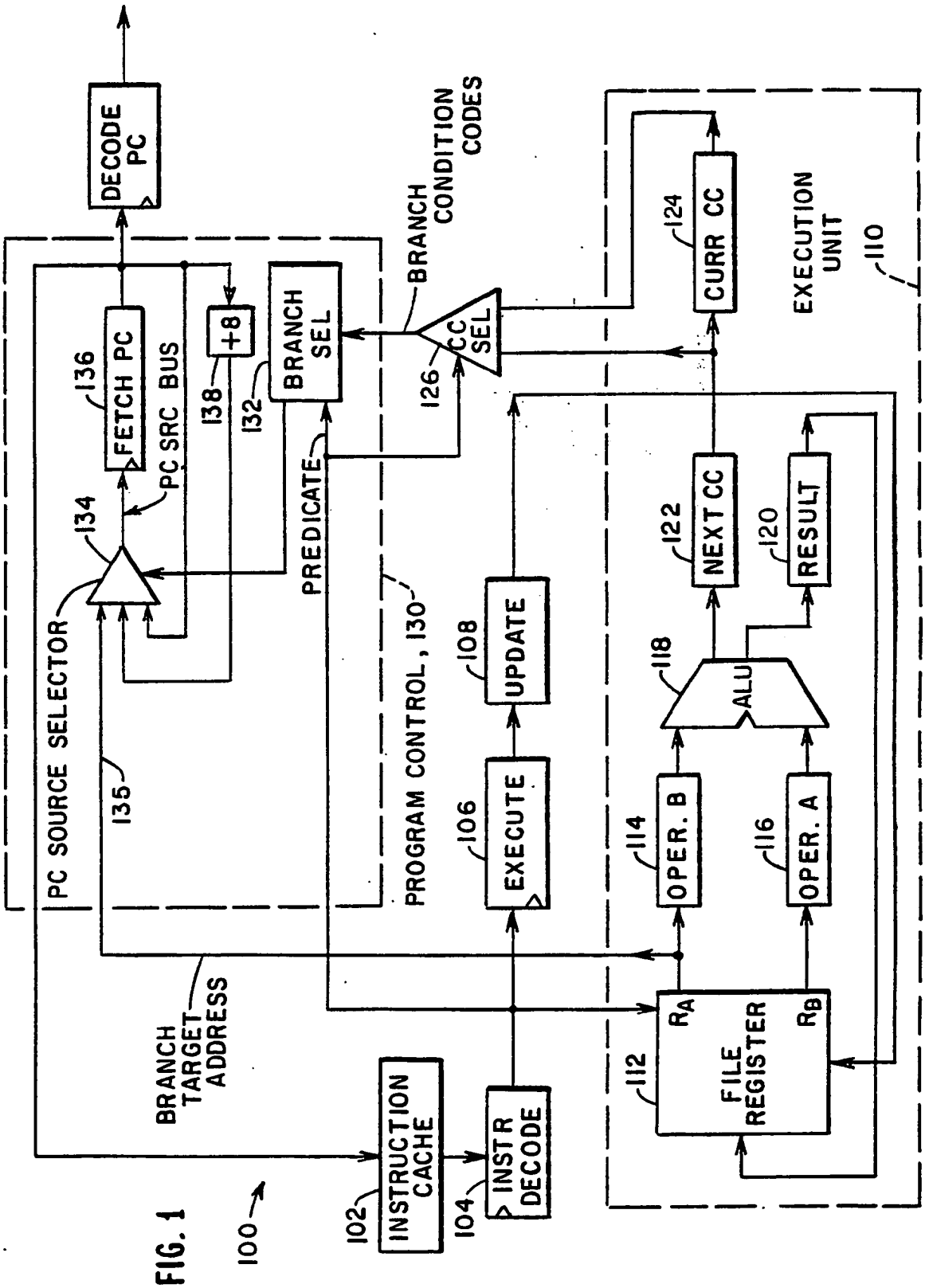


FIG. 1

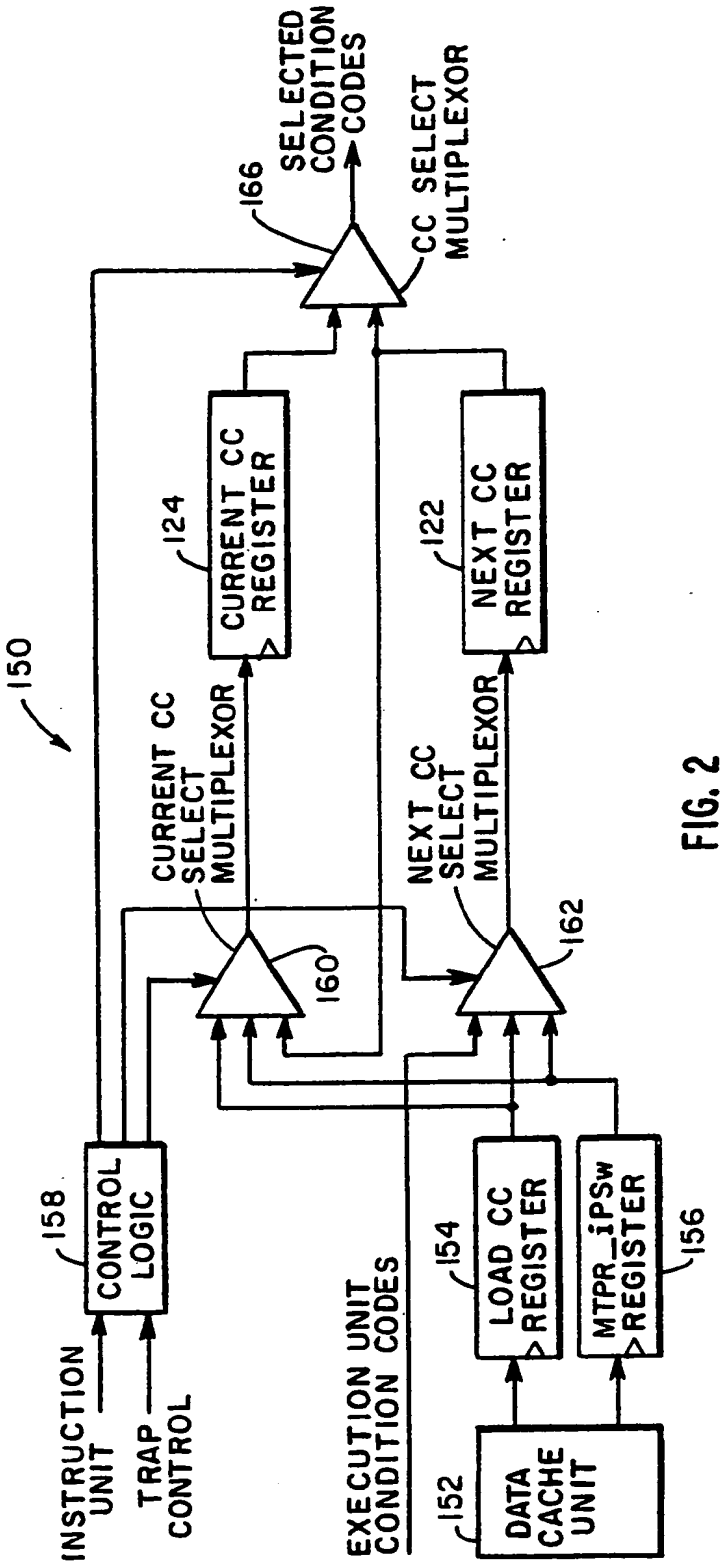


FIG. 2

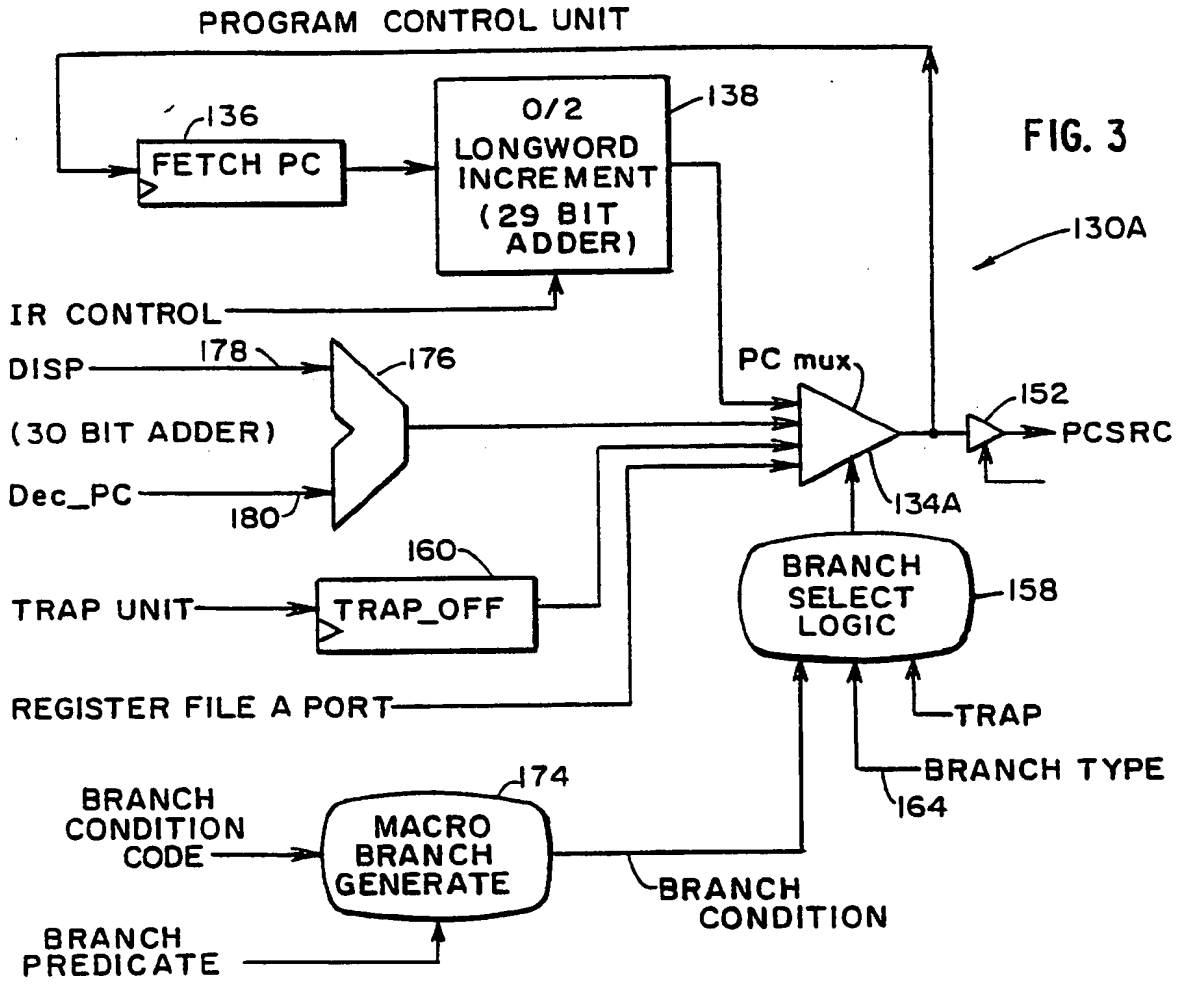


FIG. 3

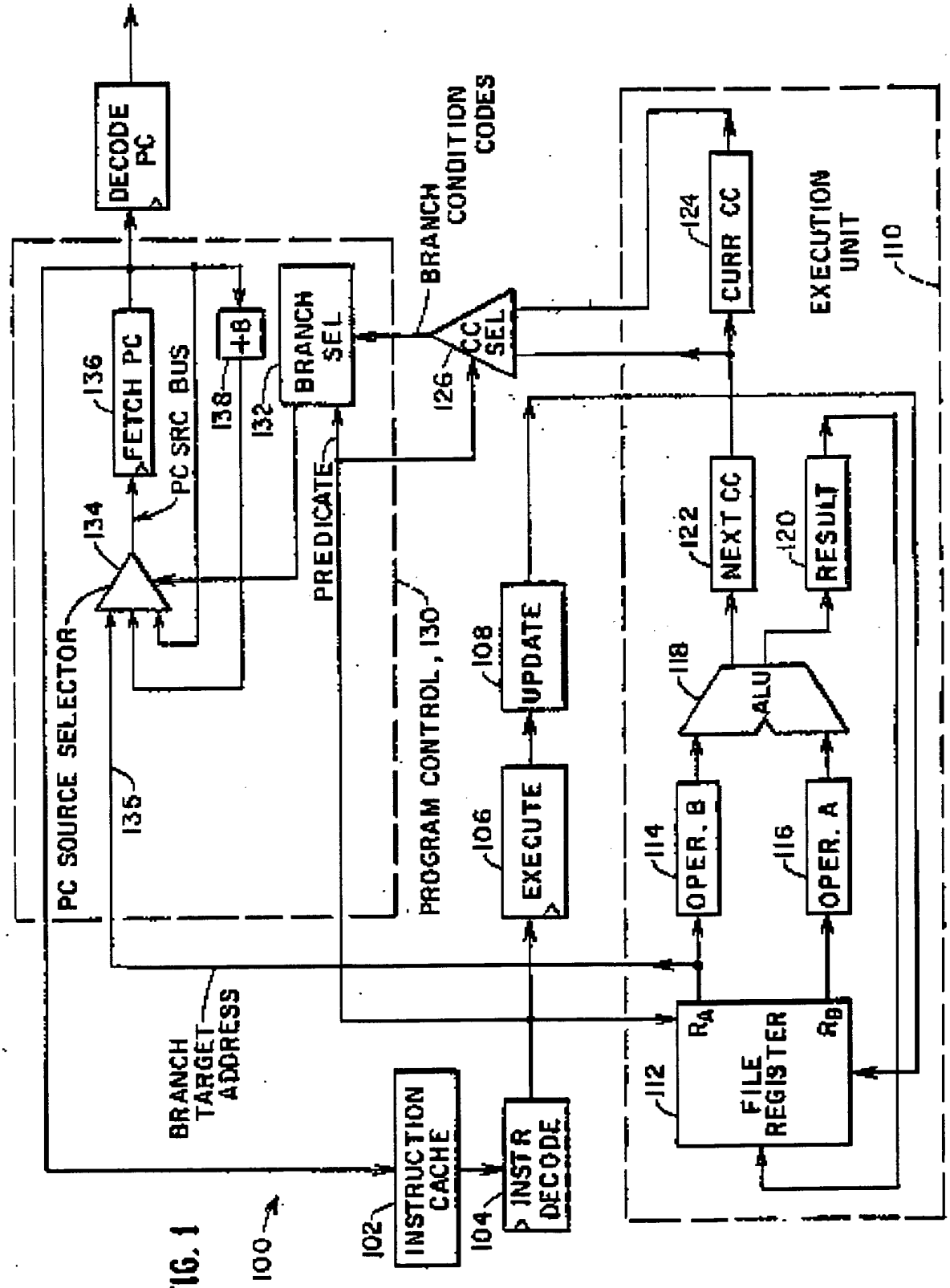


FIG. 1

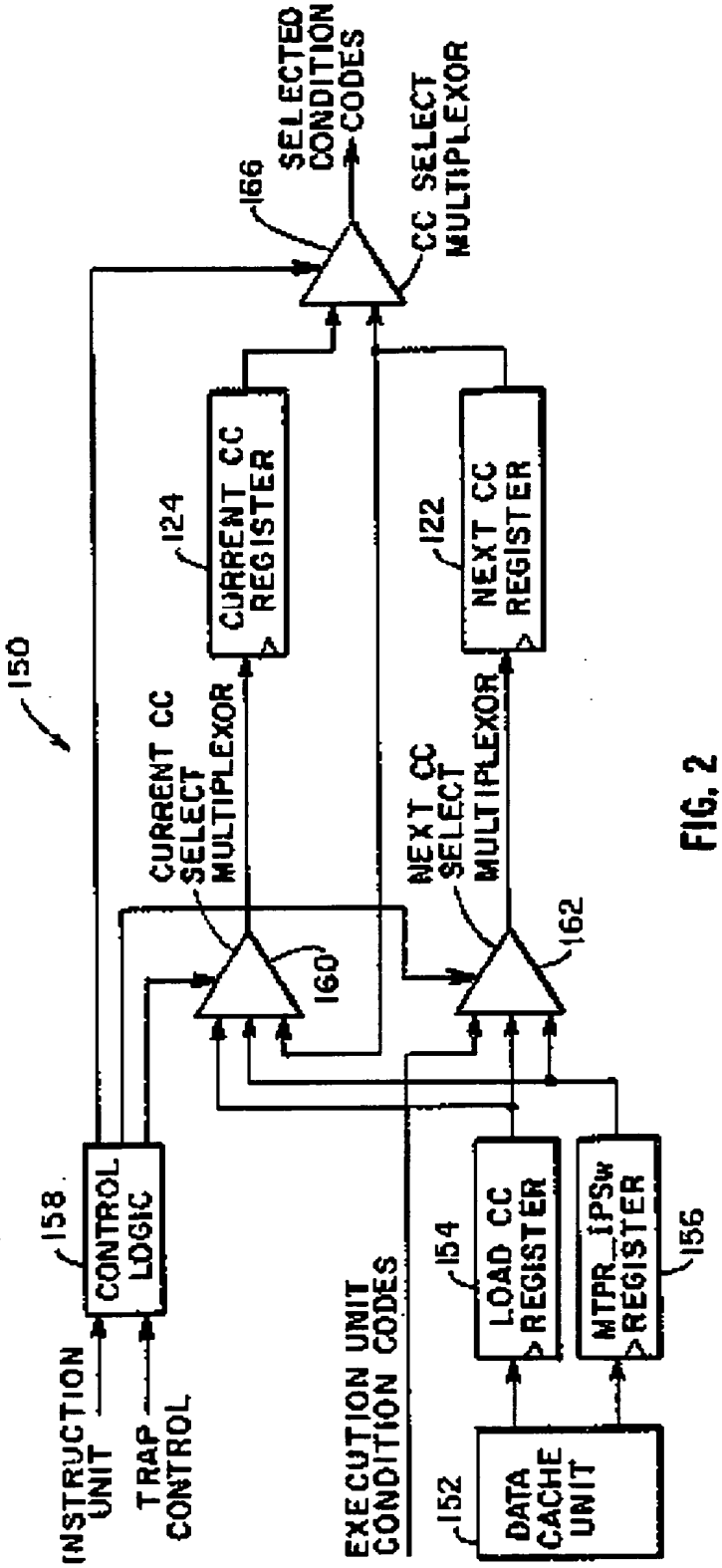


FIG. 2

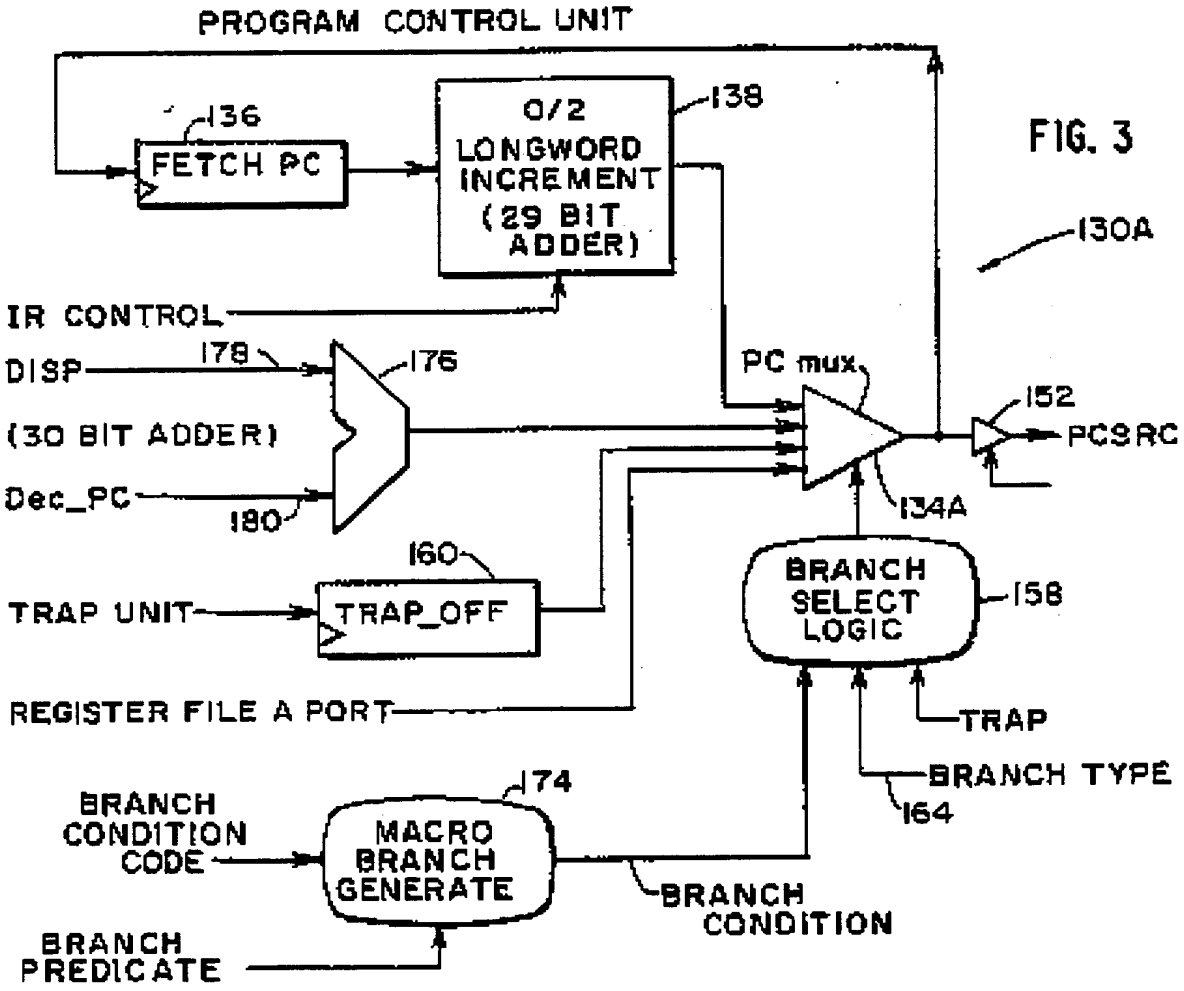


FIG. 3

(19) Europäisches Patentamt
European Patent Office
Office européen des brevets

(11) Publication number: **0 365 188 A3**

(12) **EUROPEAN PATENT APPLICATION**

(21) Application number: **89310271.5**

(51) Int. Cl.5: **G06F 9/38**, G06F 9/32

(22) Date of filing: **06.10.89**

(30) Priority: **18.10.88 US 259345**

(43) Date of publication of application:
25.04.90 Bulletin 90/17

(84) Designated Contracting States:
DE FR GB

(88) Date of deferred publication of the search report:
29.04.92 Bulletin 92/18

(71) Applicant: **APOLLO COMPUTER INC.**
33 Billerica Road
Chelmsford Massachusetts 01824(US)

(72) Inventor: **Barbour, Russell Gray**
67 Carter Drive
Framingham Massachusetts 01701(US)
Inventor: **Soeder, Carl A.**
166 Pine Hill Road
Boxborough Massachusetts 01719(US)
Inventor: **Ciavaglia, Stephen J.**
23 Macdonald Drive
Nashua New Hampshire 03062(US)

(74) Representative: **Arthur, Bryan Edward et al**
4 Dyers Buildings Holborn
London, EC1N 2JT(GB)

(54) **Central processor condition code method and apparatus.**

(57) A method and apparatus is disclosed for control of a central processor in response to a branch instruction using two separate, subsequently updated condition codes. Computer architecture is provided wherein the condition codes which determine the processor state result from the execution of instructions prior to the currently executing instruction. When the preceding instructions are executed, condition codes are set and maintained in a first condition code register. The first condition code is transferred to the second condition code register, and the first condition code register is updated to reflect the result of the current instruction execution. Any condition code state such as a branch used by the third instruction is based on the condition code state maintained in the second condition code register. The processor is provided with code which compiled according to the present invention, provides improved processor performance by reducing delays in instruction execution. If the current instruction is a condition code dependant instruction such as a branch instruction, this third instruction will execute based on the condition code maintained in the second condition code register.

EP 0 365 188 A3

FIG. 1